

Art analysis of the classification from 2024

This has been written in 2025.

The main objective is to classify the publications by themes and sub-theme.

The work from 2024

It seems to work well. However there are two possible issues:

- There are not enough themes to cover all the possible scientific publications.

Let's consider only the publications that are related to the given themes.

- The classification is only made on the *abstract* of the publication, and not on the title or even the keywords that the *HAL API* could give for instance.

I think it could be good if we gave the title, the abstract and the retrieved keywords altogether to the *classify_abstract_combined()* function.

It will give the related *theme* if the publication is within the covered range of themes.

In fact, it may be necessary to replace the *Flask API* because:

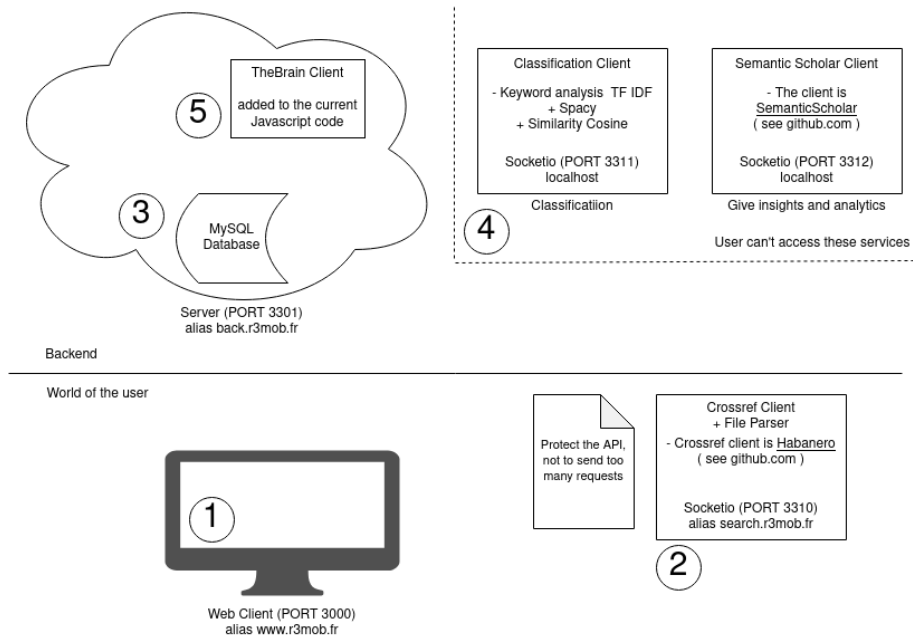
WARNING: This is a development server.
Do not use it in a production deployment.
Use a production WSGI server instead.

So anyway, the current *Flask API* will have to be reworked. Here are some ideas:

1. **Use HTTP requests with *Flask*.** This is the easiest way because it requires small changes, however this is not designed for that I suppose. In fact, the only good way to get a *Flask* server for production is to use *gunicorn*.
2. **Use *Child Processes* in Node.js.** Here is a more detailed guide *right here*.
3. **Use *Websocket* like *Socketio*.** I don't want the *websocket* to require an authentication because it will be too hard to code. That's why the client must not have the right to communicate with the *websocket*. Only the server should do that. So for this method, the server classifies the publication and not the client.

Why is it possible to use `flask_socketio` for production? Because the function `flask_socketio.SocketIO.run()` directly calls **python-gevent** or **python-eventlet** if it is available, *the doc* says.

For any of these methods, here is the generic plan on this picture:



- ① The user is connecting.
- ② The user is importing a publication.
- ③ The server receives the new publication and stores its metadata.
- ④ The server retrieves analytics and insights and then classify it.
- ⑤ The admin chooses whether the publication will go in the R3MOB brain, using the Admin page on the web client.

Figure 1: Classification Module

Better *README* file and *explanations*.

I had some issues to launch the *flask* app.

Here is a more detailed guide on how to use it for **python3.13**:

```
cd classification/code/  
python3 -m venv .env_classification  
source .env_classification/bin/activate  
pip install -r requirements.txt
```

where the *requirements.txt* file is:

```
spacy==3.8.7  
flask==3.1.1  
nltk==3.9.1  
scikit-learn==1.6.1
```

Then do:

```
python -m spacy download en_core_web_lg  
./start_classification.sh
```

If there is the error port 5000 already used, just do:

```
ps aux | grep app.py  
kill -9 <app.py pid>
```

My own thoughts about the idea

The solution is good, nothing to add, except there are not enough themes.

The different solutions that already exist.

It is possible to use a *Natural Language Processing* to get better results. However, it is not that easy.

In fact, I just need to use **BERT** which is an *NLP model* (+68M monthly downloads). It could be very powerful, more powerful than the current *classification* model.

But the current model from 2024 is good, I don't want to mess with it.

EOF